ZSW03316USEN-00

IBM Rochester Blue Gene Development
Tom Budnik (tbudnik@us.ibm.com)
Brant Knudson (bknudson@us.ibm.com)
Mark Megerian (megerian@us.ibm.com)
Sam Miller (samjmill@us.ibm.com)

1 Introduction

In V1R2 of Blue Gene/P (BG/P) there is a new software feature called High Throughput Computing (HTC). This new model of computing for BG/P enables portions of the machine to be allocated for many single-node jobs. Unlike a parallel Message Passing Interface (MPI) job, these HTC jobs or "tasks" are all independent. They start and end independently, and have no MPI communication between the tasks.

The motivation to add the HTC feature to BG/P was the recognition that there are many applications that fit well into this model of computing. They are "pleasantly parallel" in the sense that while they can run on many processors, the individual tasks can be thought of as being completely independent. For this class of applications, porting to an MPI model may involve a process of changing the code to have one or more compute nodes act as a "master node" and coordinate the work being done on the other "worker nodes".

This results in applications that have some form of the following logic:

```
int main(int argc, char *argv[])  {
 MPI_Init (&argc, &argv);
 MPI_Comm_rank (MPI_COMM_WORLD, &rank);
 if (rank == 0) {
    //  send work to other nodes and collect results
 } else {
    //  do real work
```

}

This is also called “embarrassingly parallel” by some people. Other than the MPI commands to collect the results from the other nodes, there is no MPI being done among the worker nodes.
This model can be very effective, but for some applications, it may be completely unnecessary. Prior to HTC mode becoming available, this was the only way to get a large number of independent tasks launched on Blue Gene. Since jobs are submitted to run on partitions of nodes, in order to get all of the nodes in that partition to run a job, you had to tie them all together with MPI. The applications need to dedicate one or more nodes to being the master node to achieve this.

Using the new BG/P HTC feature the master node can be shifted off a compute node and onto a front-end node.

This is a better solution for multiple reasons:

- Application resiliency: A single node failure ends the entire application in the MPI model. For HTC, only the job running on the failed node is ended while other single node jobs continue to run. For long-running jobs that require many tasks, this can mean the difference between having to start from scratch and just being able to proceed ahead on the remaining nodes. As an application scales out to more and more processing nodes this resiliency attribute becomes increasingly important because the likelihood of node failures becomes greater.

- The front-end node has more memory, better performance, and more functionality than a single compute node

- Code that runs on the compute nodes is much cleaner, since it only contains the work to be performed, and leaves the coordination to a script or scheduler. This also eliminates the need to sacrifice one node as being the master node.

- The coordinator functionality can be a Perl script, Python, compiled program, or anything that runs on Linux.

- The coordinator can interact directly with a database, to either get the inputs for the application, or to store the results. This can eliminate the need to create a flat-file input for the application, or to generate the results in an output file.

2 Making BG/P look like a Cluster

With the addition of HTC mode, Blue Gene looks more like a traditional cluster from an application's point of view. This enables a new class of workloads that use many single-node jobs and makes Blue Gene a viable solution for a wider spectrum of customers.

Blue Gene supports a hybrid application environment, mixing traditional HPC (MPI) and now HTC applications. This is possible because of the way that Blue Gene can be partitioned. Some partitions can be booted to run HPC, and others can be booted to run HTC.

By making BG/P look more like a cluster, it allows some of the basic differentiators to become major strengths when compared to traditional clusters:

- HTC leverages the low-energy, small footprint (high processor density) of a rack of 1,024 compute nodes. Blue Gene energy efficient systems hold the top 26 positions as listed by Green500 in the February 2008 rankings (http://www.green500.org/lists/2008/02/green500.php).

- Reliability: The sheer quantity of Blue Gene hardware components requires a design with a strong focus on reliability and availability. The BG/P system reliability target is less than one failure per rack per year (7 days mean time between failures (MTBF) for a 72 rack system).

- Capacity machine ("cluster buster"): In HTC mode 4,096 jobs can be run on a single rack in virtual node mode (VNM).
- Ease of Administration: Blue Gene administration is done using the powerful and easy to use web based Blue Gene Navigator.

3 Application Development

HTC applications, like HPC applications, run on the Compute Node Kernel (CNK). Application programmers see the CNK software as a Linux-like operating system. This type of operating system is accomplished on BG/P software stack by providing a standard set of runtime libraries for C, C++, and Fortran95. To the extent that is possible, the supported functions maintain open standard POSIX-compliant interfaces. An HTC application can use all the capabilities of the CNK, except MPI is not supported, so calls to Mpi_Init will fail.

See the "IBM System Blue Gene Solution: Blue Gene/P Application Development" Redbook available at www.redbooks.ibm.com for more information on the capabilities of CNK.

3.1 *submit* command

Just as ***mpirun*** is used to run a MPI job, ***submit*** is used to run a HTC job and act as a lightweight shadow for the real job running on a Blue Gene node. It is intended to simplify user interaction with the system by providing a simple common interface for launching, monitoring, and controlling HTC jobs. The ***submit*** command, much like its counterpart, ***mpirun***, is generally run from a front-end node (FEN). As a stand-alone program, ***submit*** contacts the control system to run the HTC user job. The ***submit*** command allows the user to interact with the running job via the job's standard input, standard output, and standard error.

In the framework of a scheduling system, resource allocation is coordinated by the scheduler according to the site policy. Just as ***mpirun*** is not a scheduler, ***submit*** is not a scheduler either. This means that if the user wants to run many tasks, they would want to have their scheduler be responsible for invoking ***submit*** and managing partition allocation. When used by a scheduling system, the scheduler daemons can invoke ***submit*** on behalf of the user who submitted the job. This mode of operation, called batch mode, does not allow users to interact directly with their jobs, except via predefined input and output files.

Key features:

- The ***submit*** command is simple, lightweight, and extremely fast
- Job state is integrated into MMCS, so the control system knows which nodes have jobs, and which are idle
- ***submit*** provides stdin, stdout, and stderr on a per-job basis
- Enables individual jobs to be signaled, killed or timeout
- Job meta-data (run time, exit status, etc.) is maintained in the MMCS database which allows for viewing from Navigator
- Maintains a user ID on per-job basis (allows multiple users per partition)
- Leverages support on I/O node to use an I/O daemon (CIOD) per compute node
- Designed for easy integration with job schedulers

Special considerations for the *submit* command:

The ***submit*** command provides two ways to indicate a hardware resource for a job to execute on. The first method is to use the –location keyword and specify a processor location (e.g. R01-M1-N00-J05-C00) or use a regular expression for the processor location (e.g. “R02-M1-N04-*” will run a job on any compute card on node board R02-M1-N04). The second way to indicate a hardware resource is to use the –pool parameter and specify a pool ID which will cause the job to be routed to an available processor in the compute node pool.

Example #1 (submit to location): submit -location “R00-M0-N00-J05-C00” -exe hello
Example #2 (submit to pool): submit -pool BIOLOGY –exe hello

3.1.1 Pool ID

The pool ID concept is a scheduler alias naming convention for a group of one or more partitions available to run a job on (pool ID defaults to partition name if not set). The pool ID was conceived as a simple method for a job scheduler to easily add or remove compute resources in a HTC environment. Because the pool ID is meant to be used strictly by a job scheduler and not by a Blue Gene administrator, it was intentionally excluded from the MMCS console commands that perform partition allocation and booting.

By using APIs provided by MMCS, a scheduler can create pools of nodes by grouping one or more partitions in a named pool. This is very flexible in that it allows partitions to be added and removed from pools without being rebooted. In this manner, a scheduler can dynamically adjust the number of nodes available for HTC jobs. A scheduler could have groups of users that use different pools and control the sizes of those pools. A scheduler would only have to reboot the HTC partition if they wanted to switch among the three modes (SMP, Dual, and VNM). Unlike HPC partitions, HTC partitions must have their mode determined at boot time and this cannot change from job to job without rebooting. One could envision a scheduler managing three HTC pools, one for each mode, with the rest of the machine being available for HPC jobs.

3.1.2 *submit* command syntax:

./submit [options] or ./submit [options] binary [arg1 arg2 ... argn]

Job options:

-exe <exe>	executable to run
-args "arg1 arg2 ... argn"	arguments, must be enclosed in double quotes
-env <env=value>	add an environmental for the job
-exp_env <env>	export an environmental to the job's environment
-env_all	add all current environmentals to the job's environment
-cwd <cwd>	the job's current working directory
-timeout <seconds>	number of seconds before the job is killed
-strace	run job under system call tracing

Resource options:

-mode <SMP|DUAL|VNM>	the job mode
-location <Rxx-Mx-Nxx-Jxx-Cxx>	compute core location, regular expression supported
-pool <id>	compute node pool ID

Options:

-port <port>	listen port of the submit mux to connect to (default 10246)
-trace <0-7>	tracing level, default(0)
-enable_tty_reporting	disable the default line buffering of stdin, stdout, and stderr when input (stdin) or output (stdout/stderr) is not a tty
-raise	if a job dies with a signal, ***submit*** will raise this signal

3.2 System Administration

HTC mode on BG/P is completely integrated into the Midplane Management Control System (MMCS), the control system of BG/P. Integration into MMCS means that the control system is fully aware of all HTC jobs. Those jobs are represented in the Blue Gene database when they are running and can be viewed from either the MMCS console on the service node or by using the Blue Gene Navigator from a browser. When jobs complete, information about the job is stored in the job history database table.

In BG/P, partitions can be booted to run HTC jobs and support all three operational modes: SMP, Dual, and VNM. Once the partition is booted, there is a command called ***submit*** that will run a single-node task on a partition that was booted for HTC.

A previous HTC solution for BG/L, the predecessor to BG/P, required that users provide launcher/dispatcher code to launch the tasks. Support for HTC is available out-of-the-box in V1R2, without any additional code needed.

3.2.1 MMCS Console Commands

The MMCS console provides the capability to boot blocks of compute nodes and I/O nodes. HTC jobs can't be started from the console (this must be done using the ***submit*** interface instead), but administrator commands to list and terminate running HTC jobs are available. The following section describes the important MMCS console commands for HTC:

Booting a partition to run HTC jobs:

allocate <blockId> htc=<smp|dual|vnm>

Viewing HTC jobs:

list_htc_jobs [<blockId>|*ALL [<username>]]

This command prints jobid, status, username, blockid, location, and executable for active HTC jobs.

list_jobs

The list_jobs command shows the number of HTC jobs running on a partition:

```
JOBID          STATUS USERNAME   BLOCKID     EXECUTABLE
6          Q      samjmill   R00-M0-N00  /bin/hostname

Block R00-M0-N00 has 2 HTC jobs running
Block R00-M0-N01 has 1 HTC job running
use list_htc_jobs to see HTC job details
```

Ending HTC jobs:

kill_job <jobId> [timeout]

kill_htc_jobs [block=<blockId>] [user=<username>] [timeout=<seconds>]

3.2.2 Blue Gene Navigator

The Blue Gene Navigator is an administrative tool that can be used to browse the contents of the Blue Gene database. This can be useful for problem determination. The changes to MMCS for HTC have required some changes to the Navigator interface. In many cases, the change is simply displaying that a partition has been booted in HTC mode along with its mode (SMP, VN or Dual) and pool.

The Navigator Job Summary page was originally designed to show the status of HPC jobs running on Blue Gene. In HPC mode, there are at most a couple hundred jobs running, so they can all be displayed on a single page. With the introduction of HTC, there may be thousands of jobs in the Blue Gene database. This can be overwhelming for the user and take a long time to display, so the Navigator has been enhanced to allow more filtering options and paging of the running jobs. This is shown in Figure 1. An example of how an administrator could use this is to look for a job that appears to be “stuck” by filtering on the block ID and sorting by “Status Changed” to find the oldest job.

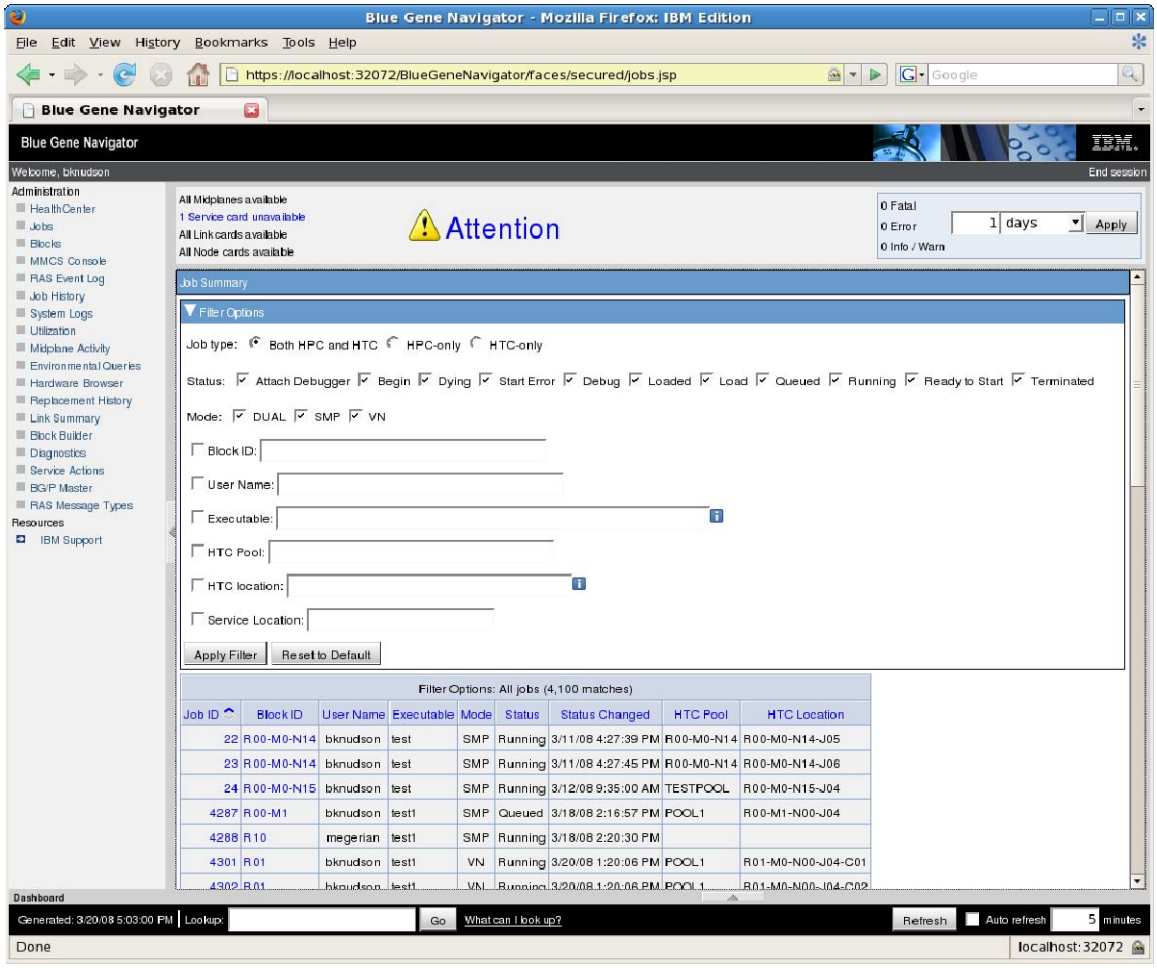

Figure 1 - Navigator Job Summary page

The Midplane Activity page in the Navigator shows a summary of the Blue Gene's midplanes and what blocks and jobs are using the hardware. This has been modified to display a summary of the HTC jobs running on the midplanes. This is shown in Figure 2. The figure shows a block "R01" that spans midplanes R01-M0 and R01-M1 that has been booted in virtual node mode for HTC in pool "POOL1". 4,095 HTC jobs are running on this rack, with 2,047 of them on R01-M0 and 2,048 on R01-M1. It also shows that the pool "POOL1" includes another midplane booted in SMP mode. Clicking on the job count for the midplane will show the Job Summary page with the filter set to display those jobs.

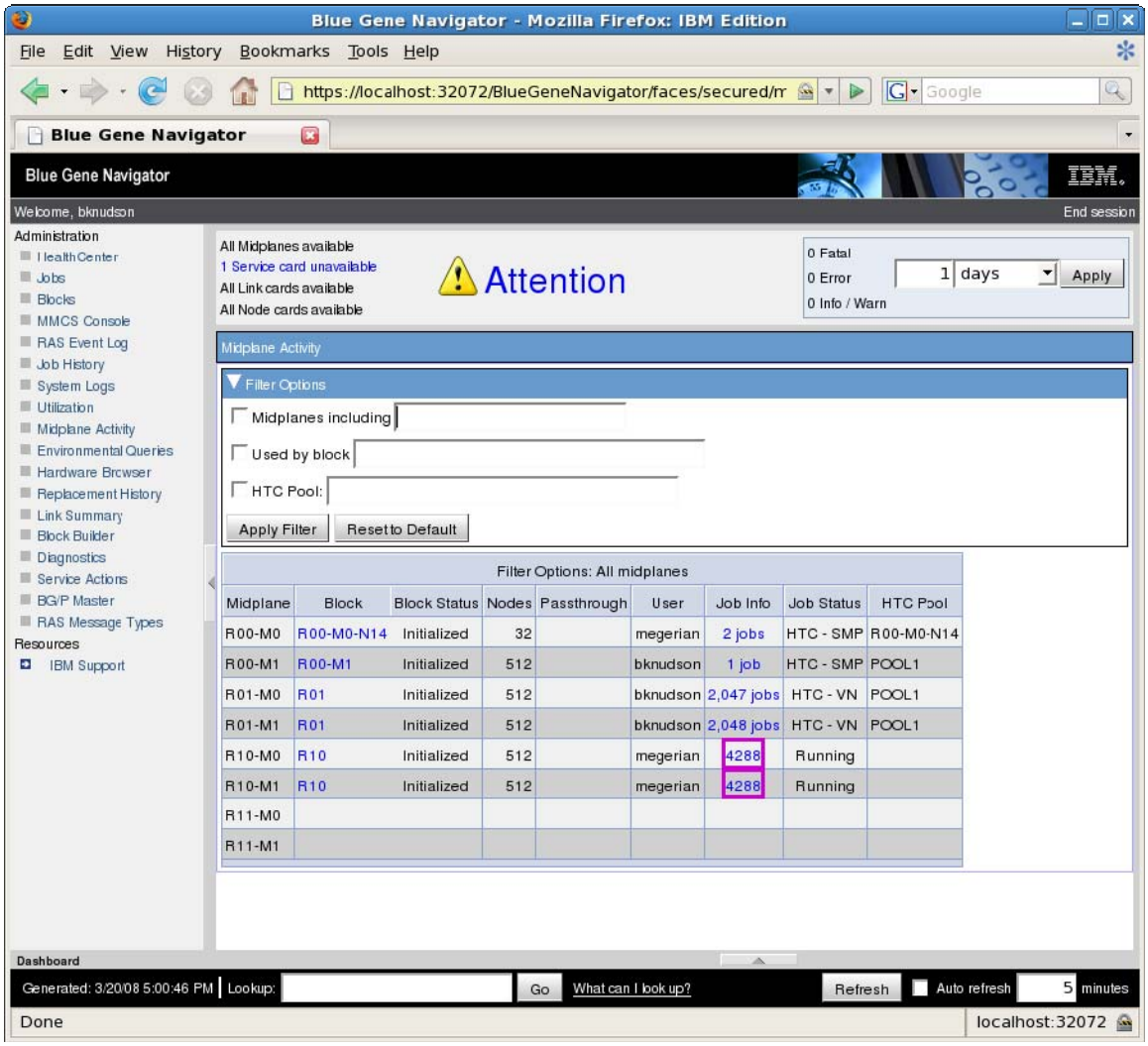

Figure 2 - Navigator Midplane Activity page

Another enhancement to the Navigator for HTC is on the Block Details page. Nodes may fail on an HTC block without causing the entire block to become unavailable. When this happens, a RAS event is generated. The Block Details page shows the locations of these types of RAS events so you can see how many failed nodes there are on a block. This is shown in Figure 3.

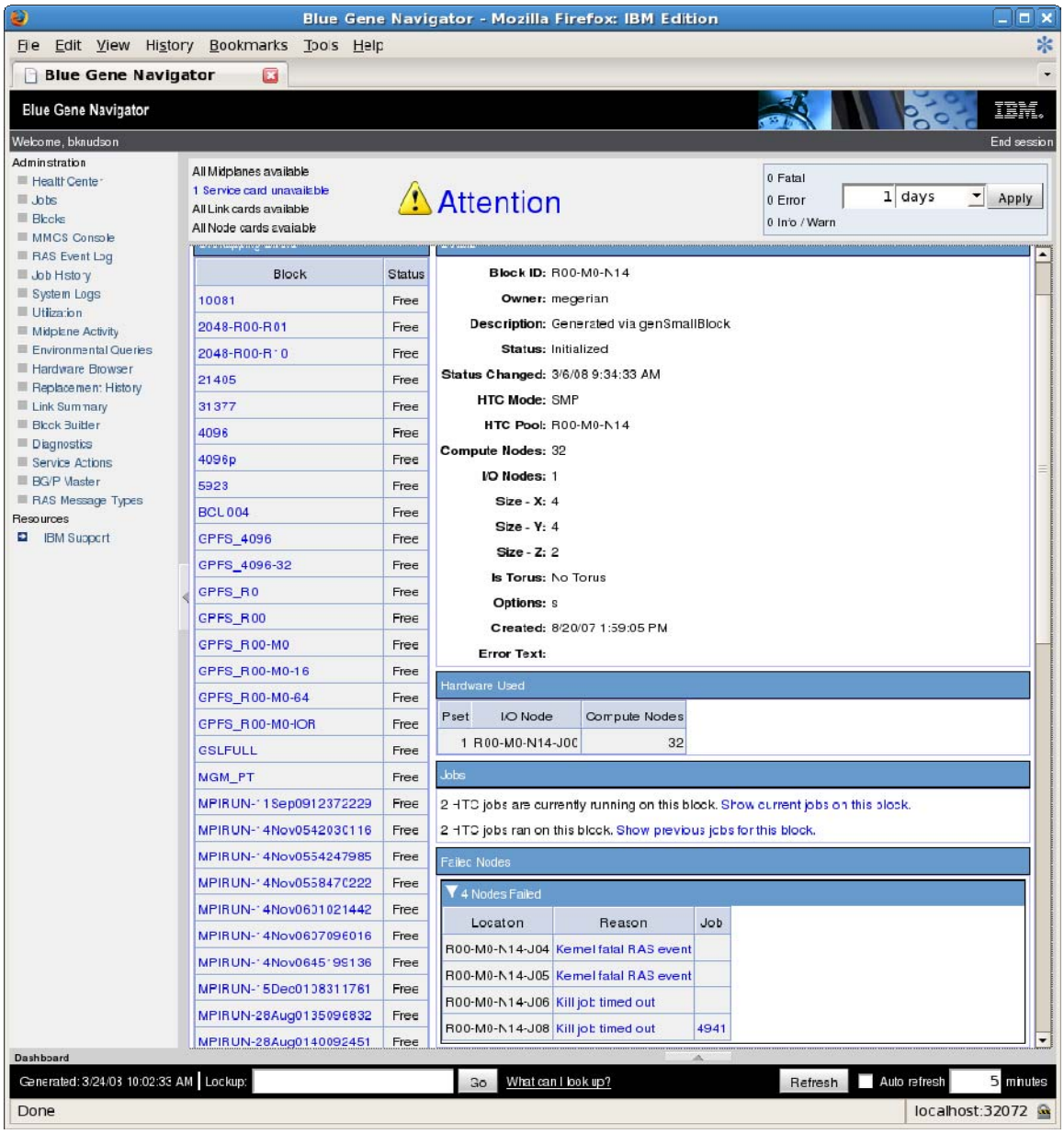

Figure 3 - Navigator Block Details

4 Underlying HTC Details

The following architecture diagram shows the interactions of each HTC component:

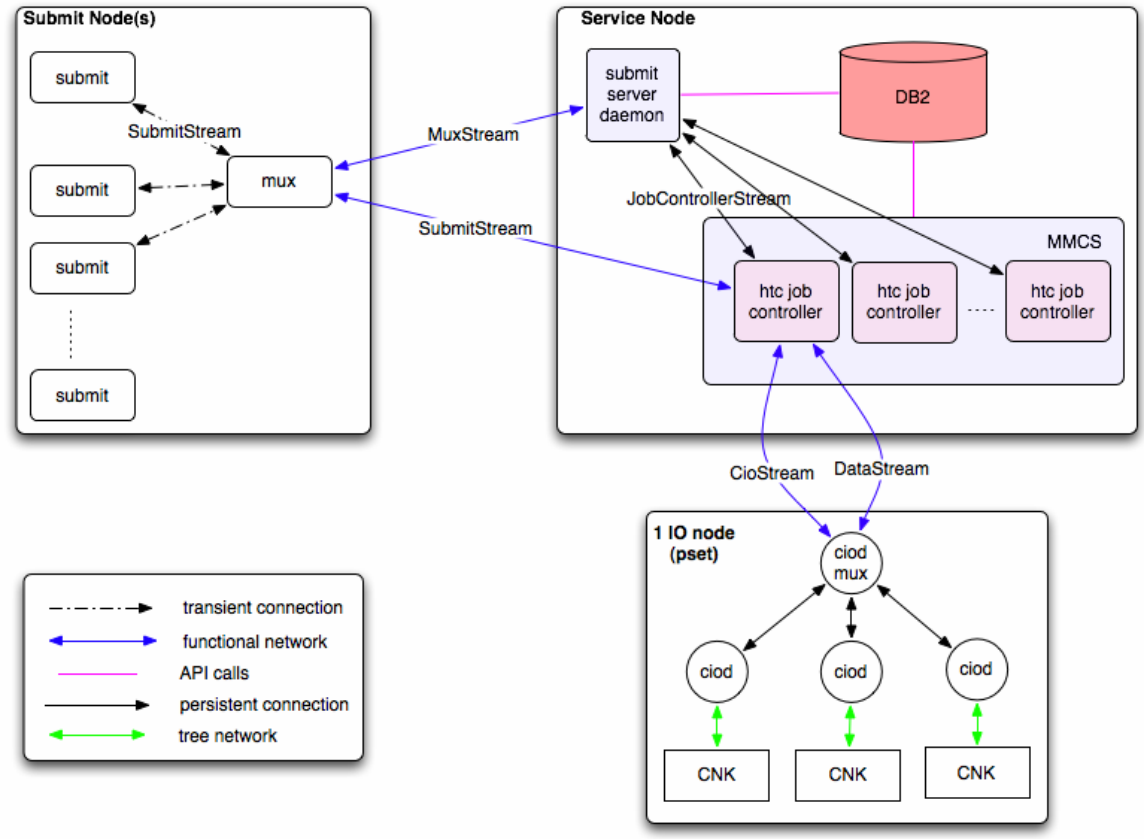

Figure 4 - Blue Gene/P HTC Architecture

4.1 Submit Node Components:

Typically a submit node is a front-end node (FEN) but the design does not limit this. A submit node could be a service node, or potentially even an IO node.

Submit Multiplexer (mux):

The submit mux federates connections from multiple ***submit*** clients together and forwards their requests to the submit server daemon and MMCS running on the service node. It is controlled with an init script rather than bgpmaster since it runs outside of the service node.

Submit:

The ***submit*** client is analogous to ***mpirun*** in the sense they both act as a shadow of the job. It transparently forwards stdin, receives stdout and stderr, and terminates when the job is complete.

4.2 Service Node Components:

Submit Server Daemon:

The submit server daemon acts as a resource arbiter for incoming ***submit*** requests. It assigns each request to an available compute node based on the pool ID, compute node location, user, and job mode. If no matches are found, an error is returned immediately rather than maintaining a queue of pending jobs.

MMCS:

The role of MMCS has not changed drastically for HTC mode. It is still responsible for booting and managing partitions and jobs, as well as maintaining machine state information in the DB2 database. Every booted HTC partition has an associated HTC Job Controller that is responsible for managing single node jobs running on the compute nodes of the partition.

Database:

HTC jobs are represented by a job ID in the job table just like their HPC counterparts. The state transitions for HTC jobs are also similar, except HTC jobs never enter the READY TO (S)TART, (D)YING, or various debugging job states. HTC jobs also appear in the job history table, with a new location column indicating the compute node location where the job ran.

4.3 IO Node Components:

CIOD Multiplexer (mux):

The CIOD mux is a new component that is only used for HTC mode. It is required because in HTC mode every compute node has a CIOD instance running on its associated IO node. Without the mux, the control system would have to open a control and data connection to each CIOD, causing it to run out of file descriptors very quickly for larger systems.

The CIOD mux has a service connection just like normal CIODs, the ***service_ciod*** MMCS console command is supported for HTC partitions.

CIOD:

When a partition is booted in HTC mode, there is one CIOD process per rank instead of one process per pset like HPC partitions.

5 Simple scheduler

HTC was built to work with any job scheduler. A scheduler developed to work in HTC mode could be very complex, as it would be designed to allocate and free partitions in HTC mode based on demand, and migrate jobs and reboot partitions when nodes have failed. For internal testing, we developed a rudimentary scheduler, dubbed the SIMPLE scheduler. Given an HTC pool that has already been booted, SIMPLE will queue up jobs and release a job for running only when there is a node available in the pool. The SIMPLE scheduler uses the submit plug-in to discover failed nodes and will attempt to restart jobs that have failed. This has been used for internal testing and development, and will be made available to Blue Gene job scheduler developers and customers.

Figure 5 shows the architecture of the SIMPLE scheduler. End-users interact with the scheduler from their workstation using the command-line tools qsub, qstat, qdel, and qcmd. The SIMPLE scheduler daemon, simple_sched, runs on the service node (or a FEN), accepting jobs to run from the command-line tools, and sending the work to the startd daemon(s) running on the submit node(s). When a startd daemon receives a job to run, it uses the ***submit*** program to run the job on a HTC node and when the ***submit*** program ends it reports the result back to the SIMPLE scheduler daemon.

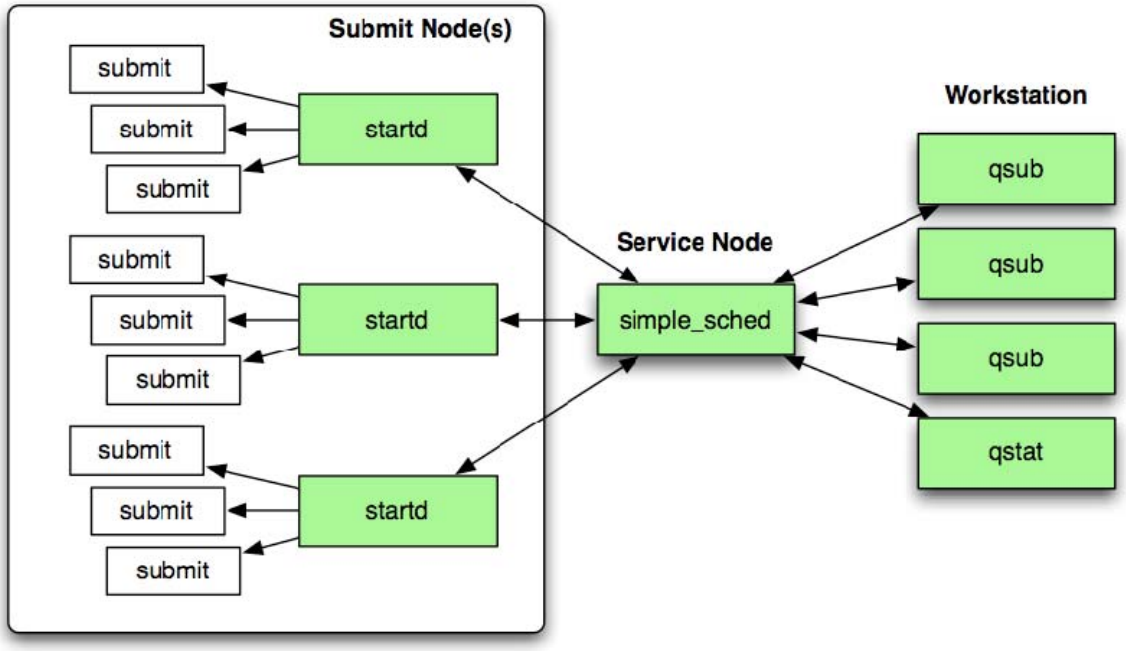

Figure 5 - SIMPLE scheduler Architecture

The SIMPLE scheduler must be given the name of the pool to use and the sizes and modes of the partitions in the pool. The partitions must already be booted

when the SIMPLE scheduler is started. The SIMPLE scheduler keeps track of how many resources are available. If a ***submit*** notifies its parent startd that the node it was running the job on has failed, the startd daemon will notify the SIMPLE scheduler daemon, which will remove that resource from the pool so that it is not used again.

The SIMPLE scheduler and the command line tools use a configuration file that contains settings like the pool name, pool size, SIMPLE scheduler hostname and port number.

The command-line tools are used by end-users or an administrator to submit jobs, retrieve the status of jobs, and control the SIMPLE scheduler daemon. The qsub command is used to submit a command to run. It returns the submit ID that the SIMPLE scheduler assigned which is then used on later requests. The options to qsub are:

./qsub [CONFIGURATION_OPTION]... [SUBMIT_OPTION]... COMMAND...

Submit options:

-cwd=DIRECTORY	Override the working directory (default is the current working directory)
-mode=MODE	The mode that the job requires
-exp_env=NAME	Export environment variable to the program
-env_all	Make all of the current environment variables available to the program
-env=NAME=VALUE[NAME=VALUE]...	Environment variable for the program
-name=NAME	Set the program name
-stdin-file=FILE	File to read stdin from (default is /dev/null)
-stdout-file=FILE	File to write stdout to (default is <name>-<submit_id>.out)
-stderr-file=FILE	File to write stderr to (default is <name>-<submit_id>.err)
-wait	Wait for results

The -mode parameter only needs to be specified when the job requires that it run in some mode (SMP, DUAL, or VN), otherwise this is omitted, indicating that the job will run in any mode. The -wait parameter indicates that qsub should wait until the job has completed. It will print status changes as they occur and exit when the job completes.

Here is an example using qsub:

```
$ qsub hello_world
Submit id: 2
```

qstat is used to get the status of a submitted job. By specifying the submit ID returned by qsub it will indicate the current status (QUEUED, ASSIGNED to a startd daemon, COMPLETED, CANCELING, CANCELED, or ERROR). Here is an example using qstat:

```
$ qstat 2
2 is ASSIGNED
$ qstat 2
2 is COMPLETED exit status 0
```

qdel is used to cancel a job. By specifying the submit ID returned by submit, if the job is still running, the ***submit*** process will be sent a SIGTERM so that it exits. Here is an example using qdel:

```
$ qsub hello_world
Submit id: 3
$ qdel 3
3 is CANCELING
$ qstat 3
3 is COMPLETED term signal 15
```

qcmd can be used to issue administrative commands. The SIMPLE scheduler can be told to not assign any more jobs using *suspend*, and to resume assigning jobs using *resume*. The status, including the number of jobs in the submit queue, and how many jobs have finished, can be retrieved by using the *scheduler_status* command.

These are the commands that can be issued from qcmd:

scheduler_status	display scheduler status
suspend	SIMPLE scheduler server will stop assigning jobs until resume
resume	SIMPLE scheduler server will resume assigning jobs
submit [SUBMIT_OPTION]... COMMAND...	submit job
status [-wait] <submitId>|all	display status for submitted job
cancel <submitId>	cancel a submitted job
help, ? [command]	display help for commands

Here is an example using qcmd to display the scheduler status:

```
$ qcmd scheduler_status
[running (submit queue=0) (submits=assigned:0 completed:3 notzero:1 error:0 canceled:0) (htc
resources=smp:0/0 dual:0/0 vn:32768/32768)]
```

qcmd will read commands from stdin if the command is not given on the command line. When this method is used, it prints out a request ID for each line. When qcmd receives the response from the server, it prints out the response. Here is an example of qcmd with no command on the command line and issuing two submits with the wait option:

```
$ qcmd
submit -wait hello_world
1 <- submit [wait]
1 -> 4 is QUEUED
1 -> 4 is ASSIGNED
submit -wait hello_world
2 <- submit [wait]
```

2 -> 5 is QUEUED
2 -> 5 is ASSIGNED
1 -| 4 is COMPLETED exit status 0
2 -| 5 is COMPLETED exit status 0

One use of the SIMPLE scheduler is to submit a batch of jobs to a single pool by a user, perhaps under the control of another scheduler that has already booted the partition. These jobs would run in parallel until they have all completed. To make this easier, a tool called ***run_simple_sched_jobs*** is provided that does the following:

1. Opens an ephemeral port to accept client connections
2. Creates a temporary configuration file based on the system configuration file using the ephemeral port number, specified pool name, and pool size
3. Starts a SIMPLE scheduler process using that configuration file
4. Starts a startd process using that configuration file
5. Starts a qcmd process using that configuration file
6. Reads programs to run from a command file, then writes “submit –wait” commands with the program to the qcmd process
7. Parses the output of the qcmd process and prints the exit status of the programs
8. Exits when all the commands have finished

These are the options when using ***run_simple_sched_jobs***:

run_simple_sched_jobs [OPTION]... [--] [COMMAND_FILE|-]...

Options:

-keep-running	Continue running after having run all commands
-base-config-file=FILE	Base SIMPLE scheduler configuration file
-config=FILE	Output personal SIMPLE scheduler configuration file
-reuse-config	Reuse configuration
-pool-name=POOL_NAME	Blue Gene HTC pool to use
-pool-size=POOL_SIZE	Size of the Blue Gene HTC pool
-simple-sched-exe=FILE	Simple scheduler executable
-startd-exe=FILE	Startd executable
-qcmd-exe=FILE	qcmd executable
-verbose[=LEVEL]	Output level
-h, -help	Print this help text

Sample output of ***run_simple_sched_jobs***:

```
$ cat cmds128.run
hello_world
hello_world
hello_world
… (128 instances of hello_world)
$ run_simple_sched_jobs -pool-name=R00-M0-N14 -pool-size=32 cmds128.run
Using temporary configuration file 'my_simple_sched.cfg.32QV0n'.
1 -| 1 is COMPLETED exit status 0
2 -| 2 is COMPLETED exit status 0
3 -| 3 is COMPLETED exit status 0
6 -| 6 is COMPLETED exit status 0
11 -| 11 is COMPLETED exit status 0
12 -| 12 is COMPLETED exit status 0
5 -| 5 is COMPLETED exit status 0
… (all 128 jobs)
123 -| 123 is COMPLETED exit status 0
126 -| 126 is COMPLETED exit status 0
127 -| 127 is COMPLETED exit status 0
125 -| 125 is COMPLETED exit status 0
128 -| 128 is COMPLETED exit status 0
Submitted 128 jobs, 128 completed, 0 had non-zero exit status, 0 requests failed.
```

6 Summary

In summary, BG/P now offers a hybrid computing model as depicted in Figure 6. Support for HTC should provide application developers the option to run exciting new workloads on Blue Gene and allow for extreme scaling of applications.

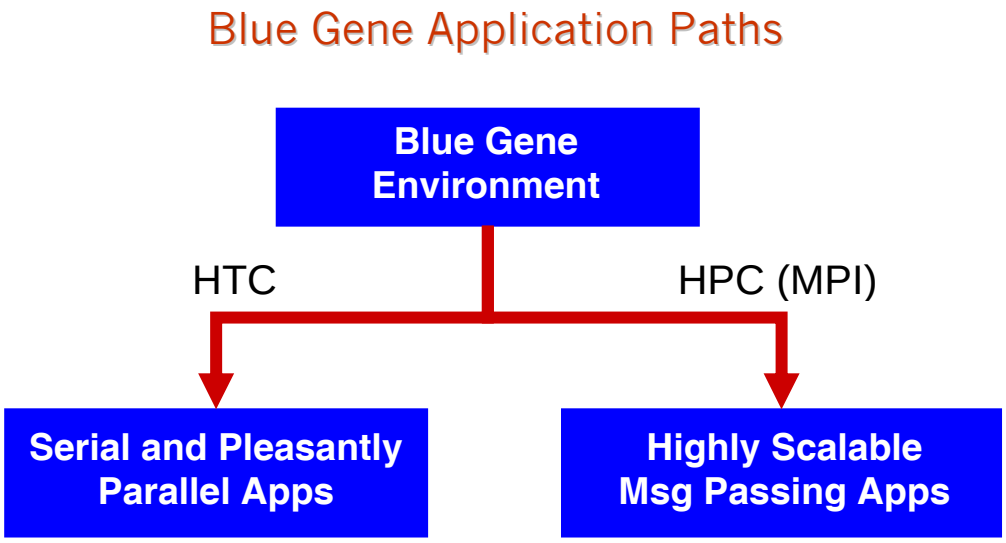

Figure 6 – Blue Gene application paths